Parallel Construction of (a, b)-trees *

Narsingh Deo Amit Jain Muralidhar Medidi

Department of Computer Science

University of Central Florida

Orlando, Florida 32816

*A preliminary version of this paper was presented at the International Conference on Parallel Processing held in August 1992 in St. Charles, Illinois.

Proposed Running Head: Parallel Construction of $(a,\ b)$-trees

Corresponding Author: Narsingh Deo
Department of Computer Science
University of Central Florida
Orlando, Florida 32816
Phone: (407) 823-2341
FAX: (407)-823-5419
email: deo@cs.ucf.edu

Abstract

We present an optimal parallel algorithm for the construction of $(a,\ b)$-trees — a generalization of 2-3 trees, 2-3-4 trees, and B-trees. We show the existence of a canonical form for $(a,\ b)$-trees, with a very regular structure, which allows us to obtain a scalable parallel algorithm for the construction of a minimum-height $(a,\ b)$-tree with N keys in $O(N/p + \log\log N)$ time using $p \leq N/\log\log N$ processors on the EREW-PRAM model, and in $O(N/p)$ time using $p \leq N$ processors on the CREW model. We show that the average memory utilization for the canonical form is at least 50 percent better than that for the worst-case and is also better than that for a random $(a,\ b)$-tree. A significant feature of the proposed parallel algorithm is that its time-complexity depends neither on a nor on b, and hence our general algorithm is superior to earlier algorithms for parallel construction of B-trees.

1 Introduction

In this paper, we present a highly parallel algorithm for the construction of height-balanced trees. Such trees are efficient data structures for supporting search, insert and delete operations needed in implementing dictionaries [9] and are typically used as index structures in databases. There are different kinds of height-balanced trees, such as AVL trees, 2-3 trees, 2-3-4 trees, B-trees, *etc*. Bayer and McCreight [1] introduced the (a, b)-tree, which provides a unified approach for all height-balanced trees. The (a, b)-tree may formally be defined as follows: Let a and b be integers with $a \geq 2$ and $b \geq 2a - 1$. A tree T is an (a, b)-tree if

1. all leaves of T have the same depth,
2. all internal nodes except the root have at least a children,
3. no node has more than b children, and
4. the root of T has at least 2 children.

An (a, b)-tree can be used as a search tree if the keys residing at a node are strictly intermediate in magnitude between the keys in the subtree to the left and those to the right. Observe that an (a, b)-tree reduces to a 2-3 tree if $a = 2$ and $b = 3$, to a 2-3-4 tree if $a = 2$ and $b = 4$, and to a B-tree if $b = 2a - 1$. For a comprehensive treatment of (a, b)-trees, see Mehlhorn [9].

The B-tree is a ubiquitous data structure used for implementing index structures in sequential computing. However, in designing concurrent-access B-trees most researchers allow nodes to be less than half-full [8, 9], and thus use the more general (a, b)-trees. Allowing b to be larger than $2a$ entails fewer restructurings in delete and insert operations, which in turn implies less blocking and higher throughput in a multiprocessor environment [17].

One of the most expensive operations on height-balanced trees is their construction. Efficient algorithms are required for construction, since such structures have to be backed up routinely and re-initialized, often to overcome the degeneration of the structure in terms of memory utilization, height *etc*. Consequently, a good deal of effort [3, 10, 13, 14] has been made in designing sequential algorithms for construction of 2-3 trees and B-trees. Wang and Chen [15] presented optimal algorithms for constructing 2-3 trees in parallel from a sorted list of data items, or keys. Wang, Chen and Yu [16] recently extended the Wang and Chen's results to B-trees, or (a, $2a-1$)-trees. But, their B-tree construction algorithm is not independent of the value a for the tree. For example, their algorithm would not be as efficient for constructing a (100, 199)-tree as it would be for a (4, 7)-tree, since the processor allocation is one per node. In fact, the algorithm has a time complexity of $O(a + \log\log N)$ and can use only $O(N/a)$ processors to construct any practical-size B-tree with N keys. It should also be noted that Moitra and Iyengar [11] as well as Chang and Iyengar [5] have addressed the problem of constructing complete binary search trees on the EREW PRAM model. Dekel, Peng and Iyengar [6] extended this technique to construct m-way search trees. But, these binary and m-way search trees are not well suited for insertions and deletions required in dynamic data structures.

A search tree with rigid structure, such as a complete binary tree, can be efficiently constructed in parallel using the shape of the tree for a given number of keys. As we move from binary and m-way search trees to less constrained 2-3 trees, B-trees, and finally to the least constrained (a, b)-trees, the structure of the tree is less predictable, making fast construction in parallel harder.

In Section 2 we first show the existence of a canonical (a, b)-tree, the first of its kind with a very regular structure and minimum height, which enables us to devise a fast parallel algorithm for its construction. We show that on the average the memory utilization for the canonical (a, b)-trees is at least 50 percent better than the worst-case utilization. Even when restricted to B-trees, our canonical form has better memory utilization than the tree constructed by Wang, Chen and Yu's algorithm [16]. In Section 3 some properties of the canonical $(a,\ b)$-tree are derived, which are useful in its construction. The construction algorithm is then described in Section 4. Our algorithm explicitly takes care of processor scheduling such that the time-complexity is independent of both a and b. The time-complexity of our algorithm on the EREW PRAM is shown to be $O(\log\log N)$, which puts this problem in the set of the remarkably few problems known to have a very fast (i.e. doubly-logarithmic time) and optimal parallel algorithm [2]. Finally, some further research directions are discussed in the concluding section.

2 Canonical $(a,\ b)$-trees

In an $(a,\ b)$-tree, any internal node with q children contains $(q-1)$ keys and will be denoted as a *q-node*, where $a \leq q \leq b$. In the rest of the paper, we number the levels from 1 (the root level) to h (the last level). Observe that the minimum number of levels in an $(a,\ b)$-tree with N keys is $h = \lceil \log_b(N+1) \rceil$. The existence of a canonical minimum-height $(a,\ b)$-tree is shown next.

2.1 Existence

The canonical form is a minimum-height $(a,\ b)$-tree in which (1) level 1 to level $c-1$ contain only *a-nodes*, (2) level c to level $h-1$ contain only *b-nodes*, and (3) at the bottom level h the first l leaves, except for the leftmost one, are *b-nodes* and the rest are *a-nodes*. The leftmost leaf is a *q-node* where $a \leq q \leq b$. Figure 1 shows a schematic for the canonical $(a,\ b)$-tree. The leftmost leaf, depicted as Region III, is the *q-node*.

Let us assume that the leftmost leaf is a *b-node* for a simpler analysis. Then the total number of keys in this $(a,\ b)$-tree is:

$$N = a^c b^{h-c} - 1 + l(b-a). \tag{1}$$

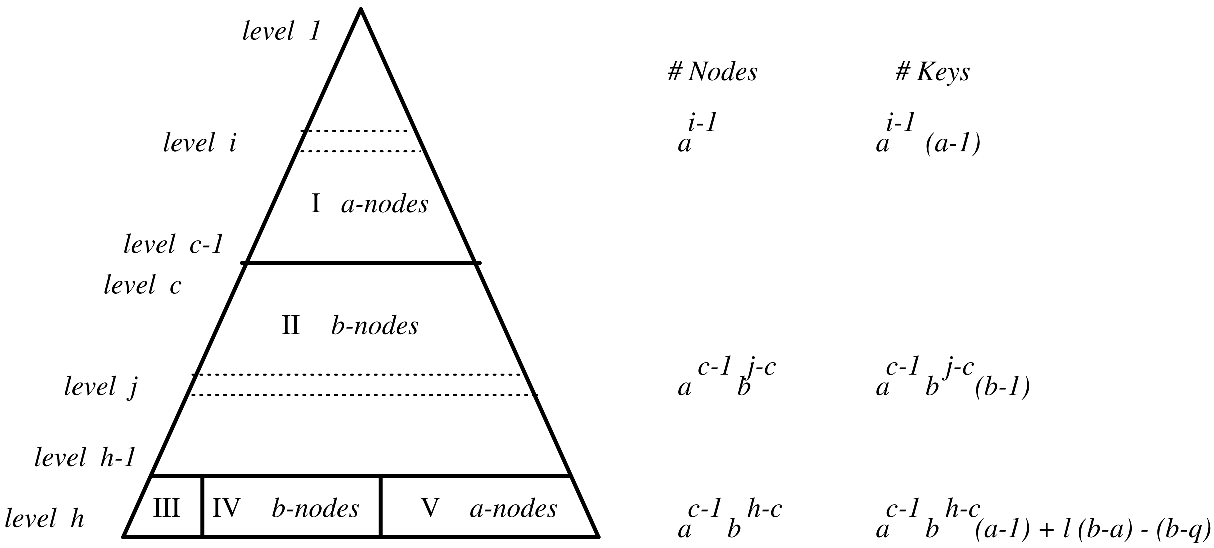

Figure 1: Canonical $(a,\, b)$-tree

The obvious constraints on integers c and l are:

$$1 \leq \quad c \leq \quad h, \text{ and} \tag{2}$$

$$1 \leq \quad l \leq \quad a^{c-1}b^{h-c}. \tag{3}$$

The constraint on l implies that there is at least one *b-node* at the leaf level. The maximum value for l is the number of leaves, $a^{c-1}b^{h-c}$ in the canonical form. From eqs. (1) and (3), we obtain:

$$c - 1 \leq \log_{(b/a)} \frac{b^h}{N+1} < c.$$

Hence $\lfloor \log_{(b/a)}(b^h/(N+1)) \rfloor + 1$ is the unique integral solution for parameter c in the preceding inequality. This solution must also satisfy eq. (2). For $c = 1$ and $h = 1$, the tree has only one node, and $l = 1$ implies that it must be a *b-node*, and hence $N + 1 \geq b$ for the canonical form. The inequality $c \leq h$ is satisfied if $\log_{(b/a)}(b^h/(N+1)) < h$. Hence for any N such that $a^h < N + 1 \leq b^h$ and $h \geq 2$ a unique solution of the required shape exists, assuming that N fits the form of eq. (1), which implies:

$$N - (a^h - 1) \equiv 0 \mod (b - a) \tag{4}$$

The solution can be extended to all N in the range $a^h < N + 1 \leq b^h$. If the given N does not satisfy eq. (4) then a minimum $N' \geq N$ that satisfies eq. (4) is:

$$N' = N + \alpha \;\text{ where }\; \alpha = (b - a) - ((N - (a^h - 1)) \bmod (b - a))$$

Note that $h = \lceil \log_b(N' + 1) \rceil = \lceil \log_b(N + 1) \rceil$, since $N' = b^h - 1$ is a solution that satisfies eq. (4). A tree with N' keys can be reduced to one with N keys by removing α keys from the leftmost leaf. The leftmost leaf is a *b-node* and $1 \leq \alpha \leq b - a - 1$, hence the *b-node* changes into one with at least a keys. Nevertheless, the tree shape remains exactly the same.

Thus, there exists a uniquely shaped tree as long as $a^h < N + 1 \leq b^h$ for $h \geq 1$. However there are intervals, such as $(b, a^2], (b^2, a^3], \ldots$, where $N + 1$ does not fall in this range. It can be shown that the number of these intervals is finite and their total size is independent of N as follows.

Lemma 2.1 *For all $N + 1 > a^k$, where $k = \lfloor \log_{(b/a)} b \rfloor$, there exists a unique $(a,\ b)$-tree of the required canonical form.*

Even though the threshold a^k is fixed, it could be quite large when a and b are large. But even in the range $b^{h-1} < N + 1 \leq a^h$, a similar canonical $(a,\ b)$-tree can be shown to exist.

2.2 Canonical Shallow $(a,\ b)$-trees

In the canonical shallow $(a,\ b)$-tree, the root needs to be a *d-node* where $2 \leq d < a$. The remaining $N - d + 1$ keys are split into d partitions, where the size of each partition is either $\lfloor (N - d + 1)/d \rfloor$ or $\lceil (N - d + 1)/d \rceil$. For each of these partitions a unique $(a,\ b)$-tree can be constructed as described earlier, if the following inequality is satisfied.

$$a^{h-1} < \left\lfloor \frac{N-d+1}{d} \right\rfloor + 1 \leq \left\lceil \frac{N-d+1}{d} \right\rceil + 1 \leq b^{h-1}. \qquad (5)$$

Inequality (5) can be shown to hold by first showing that it holds for the base case when $N+1 = b^{h-1}+1$ by letting $d = 2$. Then an inductive argument can be used to show that as $N+1$ ranges up to a^h there exists some value of d such that eq. (5) is true. Hence, we have the following.

Theorem 2.2 *For all $N \geq 1$ there exists a unique $(a,\ b)$-tree with the required canonical form.*

2.3 Memory Utilization

In a typical $(a,\ b)$-tree implementation, each node is allocated a fixed-size block of memory capable of holding a maximum of b pointers and $b-1$ keys. Thus for a given number of keys, an $(a,\ b)$-tree with fewer nodes implies less wasted space and consequently better memory utilization.

Let V_N be the number of nodes in the canonical $(a,\ b)$-tree for N keys. Since any $(a,\ b)$-tree with N keys has at least $N/(b-1)$ nodes, memory utilization can be expressed as $N/((b-1)V_N)$. No non-trivial lower bound is known for the minimum number of nodes in an $(a,\ b)$-tree. The worst-case memory utilization of any $(a,\ b)$-tree is $(a-1)/(b-1)$.

In the canonical form, there are instances where the memory utilization is 1.0, which is the best possible, and also instances where the utilization is as low as $(a-1)/(b-1)$. Since V_N can be computed, the memory utilization can be determined for any N, the number of keys. Unfortunately the utilization does not approach a limit as $N \to \infty$. Similar to Bitner and Huang [3], we would prefer a simple numerical estimate of how these trees perform on the average. Thus a meaningful measure for the canonical form is the *average utilization* over all values of N for given a and b.

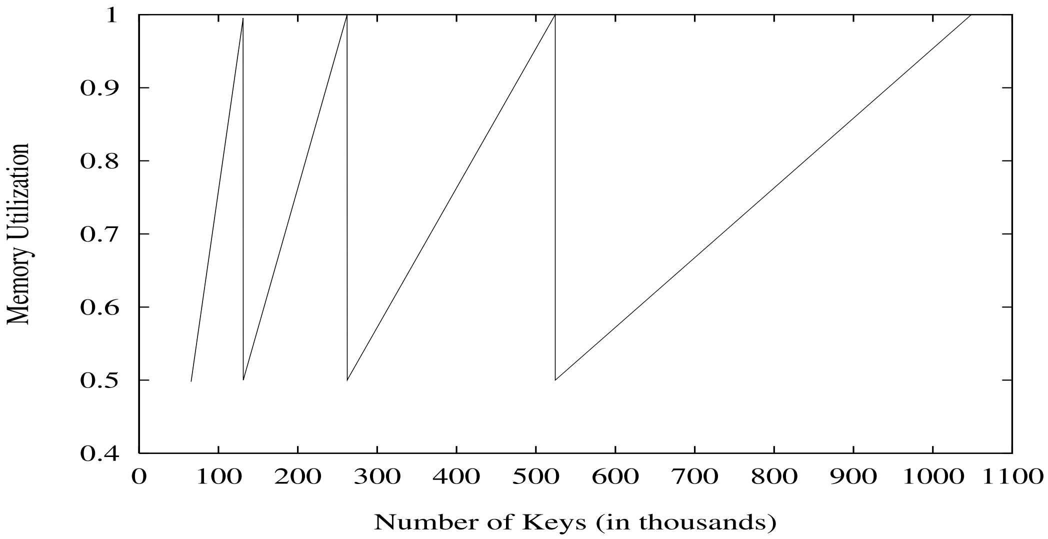

Figure 2: Memory utilization of canonical (8,16)-trees with 5 levels

Consider Figure 2, in which the memory utilization of an (8,16)-tree is plotted as N is varied from 16^4 to $16^5 - 1$, that is, for all canonical trees with 5 levels. The same pattern is repeated for any interval $[16^{h-1}, 16^h)$, where h is the number of levels. The average memory utilization in the interval shown is approximately 0.75, which is the average utilization for canonical (8,16)-trees as the same pattern repeats in all intervals.

In general, for any $(a,\ b)$-tree, the average memory utilization can be computed similarly. Since the number of shallow canonical $(a,\ b)$-trees is fixed, we can ignore them for the purpose of computing the average utilization for large values of N. In any interval $[b^{h-1}, b^h)$, the value of c varies from 1 to $\lfloor \log_{b/a} b^h/(b^{h-1}+1) \rfloor + 1$, which is bounded by $\lfloor \log_{b/a} b \rfloor + 1$. For a fixed value of c, in this interval, the number of nodes in the tree is fixed. Hence as the number of keys increases, the memory utilization linearly increases from $(a-1)/(b-1)$ (when the number of *b-nodes* in the leaf level is zero) to almost 1.0 (when all the leaves are *b-nodes*). For $c = 1$ the utilization would reach to the value 1.0, but for other values of c there are some *a-nodes* in the levels 1 through $c-1$ and hence the utilization is almost, but not exactly, 1.0. Since the maximum value of c is independent of N for a given a and b, we can ignore the wastage of memory in these few nodes for calculating the average utilization. Hence for each value of c the average utilization is $(a+b-2)/(2b-2)$, which is also the average utilization of the interval as well as that of the canonical form. This is significantly better than the worst case memory utilization of $(a-1)/(b-1)$. Mehlhorn [9] estimates the memory utilization of a random $(a, 2a)$-tree to be 0.69 whereas the average utilization of our canonical form for $b = 2a$ is about 0.75. Similarly our canonical form, for B-trees, has better memory utilization than the one constructed by Wang, Chen and Yu [16]. For example, for a 2-3-4 tree the ratio of the number of nodes in their tree versus that in our canonical form can be as much as 5/4 as the height increases [7].

3 Properties of the canonical $(a,\ b)$-tree

In this section, we introduce some properties of the canonical $(a,\ b)$-tree T_N, described in Section 2.1. Similar properties can be obtained for the shallow $(a,\ b)$-trees, characterized in Section 2.2. For the convenience of description, the following notations will be used in the rest of the paper.

- $\langle i, j\rangle$: The jth node (from the left) in the ith level of the tree.
- $(\langle i, j\rangle, k)$: The kth key in the node $\langle i, j\rangle$.
- $\mathrm{CHILD}_N(\langle i, j\rangle, k)$: The kth child of node $\langle i, j\rangle$ in T_N. (If node $\langle i, j\rangle$ has x keys, $\mathrm{CHILD}_N(\langle i, j\rangle, k)$ is undefined for all $k > x + 1$.)
- $\mathrm{RANK}_N(\langle i, j\rangle, k)$: The rank of the kth key in node $\langle i, j\rangle$ in the sorted list, which is the number of keys (inclusive of the key) that precede the $(\langle i, j\rangle, k)$ key in the inorder traversal of T_N. (If node $\langle i, j\rangle$ has only x keys, $\mathrm{RANK}_N(\langle i, j\rangle, k)$ is undefined for all $k > x$.)

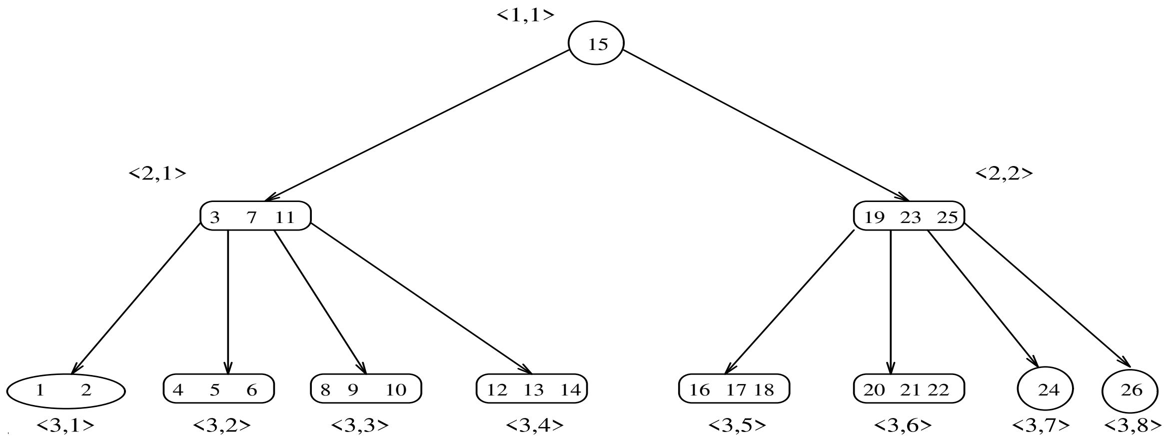

Figure 3: An 2-3-4 tree with node labeling

- $\text{LEAF}_N(\langle i, j\rangle, k)$: The number of leaf nodes in T_N that precede the key $(\langle i, j\rangle, k)$ in the inorder traversal. (If node $\langle i, j\rangle$ has only x keys, $\text{LEAF}_N(\langle i, j\rangle, k)$ is undefined for all $k > x$.)

Figure 3 shows the node labeling $\langle i, j\rangle$ for a 2-3-4 tree, with $N = 26$. Consider the key 19, the first one in the node $\langle 2, 2\rangle$, whose rank is 19. The number of leaf nodes that precede this key is $\text{LEAF}_{26}(\langle 2, 2\rangle, 1) = 5$. Also the first child of the node $\langle 2, 2\rangle$ is $\text{CHILD}_{26}(\langle 2, 2\rangle, 1) = \langle 3, 5\rangle$, the second child is $\text{CHILD}_{26}(\langle 2, 2\rangle, 2) = \langle 3, 6\rangle$, and so on.

The shape of the constructed $(a,\ b)$-tree is uniquely determined by the values of h and c. From the definition of this tree, the following lemma can be derived.

Lemma 3.1 *For each node $\langle i, j\rangle$ in T_N,*

(a) when $1 \le i < c$, CHILD$_N(\langle i, j\rangle, k) = \langle i + 1, a(j - 1) + k\rangle$, where $1 \le k \le a$ and is undefined for other values of k.

(b) when $c \le i < h$, CHILD$_N(\langle i, j\rangle, k) = \langle i + 1, b(j - 1) + k\rangle$, where $1 \le k \le b$ and is undefined for other values of k.

(c) when $i = h$, CHILD$_N(\langle i, j\rangle, k)$ is undefined for all k.

For any node $\langle i, j \rangle$ in T_N, $1 \leq i < c$, the leaf nodes that precede the kth key in the inorder traversal are the leaves contained in the subtrees rooted at $\langle i{+}1, 1\rangle, \langle i{+}1, 2\rangle, \ldots, \langle i{+}1, a(j{-}1){+}1\rangle$, $\ldots, \langle i+1, a(j-1)+k\rangle$ nodes. If $c \leq i \leq h$, the leaf nodes that precede the kth key in the $\langle i, j\rangle$ node can be computed similarly yielding the following.

Lemma 3.2 *$LEAF_N(\langle i, j\rangle, k)$, whenever defined, equals $(a(j-1)+k)a^{c-i-1}b^{h-c}$ if $1 \leq i < c$, $(b(j-1)+k)b^{h-i-1}$ if $c \leq i < h$, and $j-1$ if $i = h$.*

It is possible to obtain the same shaped canonical tree for different values of N. For simplicity in introducing the properties of T_N, we use a same shaped tree with the maximum number of keys possible. This tree can be obtained by making all the leaves *b-nodes* and is denoted by $\mathrm{T}_{N_{max}}$. Observe that the subtrees rooted at any node in a given level contain the same number of keys in $T_{N_{max}}$.

Lemma 3.3 *In $T_{N_{max}}$, the subtree with root at node $\langle i, j\rangle$ contains $a^{c-i}b^{h-c+1}-1$ keys if $1 \leq i < c$ and $b^{h-i+1}-1$ keys if $c \leq i \leq h$.*

Proof. Follows from the regular structure of the tree $\mathrm{T}_{N_{max}}$. $\square$

The keys in between the $(k-1)$th and the kth keys of the node $\langle i, 1\rangle$, while traversing $\mathrm{T}_{N_{max}}$ in the inorder traversal, are the ones in the kth subtree of node $\langle i, 1\rangle$ and hence Lemma 3.3 leads to the following.

Corollary 3.4 *$RANK_{N_{max}}(\langle i, 1\rangle, k)$ equals $ka^{c-i-1}b^{h-c+1}$ when $2 \leq k < a$ and $1 \leq i < c$, and kb^{h-i} when $2 \leq k < b$ and $c \leq i \leq h$.*

Moreover, there exists exactly one key between any two consecutive nodes in a level, in their lowest common ancestor, that separates them. Hence, we have the following.

Lemma 3.5 *RANK*$_{N_{max}}(\langle i,j\rangle ,k) = (j-1)a^{c-i}b^{h-c+1} + ka^{c-i-1}b^{h-c+1}$ *if* $1 \leq i < c$ *and* $1 \leq k < a$, *and* $(j-1)b^{h-i+1} + kb^{h-i}$ *if* $c \leq i \leq h$ *and* $1 \leq k < b$.

In the $(a,\ b)$-tree T_N constructed for arbitrary N, the first leaf node has α keys fewer than in $T_{N_{max}}$ and only leaves 2 through l are *b-nodes*. Accounting for these differences and using the previous lemmas gives the main result of this section:

Theorem 3.6 *RANK*$_N(\langle i,j\rangle ,k)$*, whenever defined, equals k if i = h, j = 1 and* $1 \leq k \leq b-1-\alpha$*, and RANK*$_{N_{max}}(\langle i,j\rangle ,k)$ *- (b-a)(max{ 0, LEAF*$_N(\langle i,j\rangle ,k)$ *- l }) -* α *otherwise.*

4 Parallel Construction Algorithm

In this section we present a parallel algorithm for constructing the canonical $(a,\ b)$-tree T_N from a sorted list of keys. Wang and Chen's [15] parallel construction algorithm for 2-3 trees assigns nodes to a processor. A similar strategy to construct $(a,\ b)$-trees would lead to an $O(b + \log\log N)$ algorithm on an EREW PRAM model. Even for moderate values of b, the time complexity is not dominated by $\log\log N$ but by b, unless N is impracticably large. Ideally, we would like to have an algorithm for which time complexity is independent of a and b. To achieve such an algorithm, each processor will be assigned a set of keys rather than a set of nodes, which complicates the processor allocation problem.

Let A_N be a linear array, of size $(2b-1)V_N$, representing the $(a,\ b)$-tree. We assume that the nodes of the tree are to be stored in the linear array A_N in breadth-first order, just as in [15]. Each node is allocated $2b-1$ consecutive locations: $b-1$ for the keys and b for the child pointers. Hence, node $\langle i,j\rangle$ of T_N will have the index $(a^{i-1}-1)/(a-1)+j$, if $1 \leq i \leq c$, and $(a^{c-1}-1)/(a-1)+a^{c-1}(b^{i-c}-1)/(b-1)+j$, if $c < i \leq h$, in A_N.

Before presenting the optimal parallel algorithm, let us first consider the case of constructing the tree T_N given N processors under the EREW PRAM. Each processor is assigned one key to fill in the tree T_N. The processor P_r finds, for the rth key in breadth-first ordering of T_N, the node $\langle i, j\rangle$ to which r belongs and the offset k. In order to find $(\langle i, j\rangle, k)$ each processor can initially find the index r' of the node $\langle i, j\rangle$ to which the assigned key belongs and also the offset, k, within that node. Consider the $(a,\ b)$-tree schematic shown in Figure 1 where five distinct regions are identified. The index of the node containing the key and the offset can be computed by considering these regions separately. For example, consider Region II where $a^{c-1} \leq r < a^{c-1}b^{h-c}$. Let s_a be the total number of nodes in the levels 1 through $c-1$ (Region I in Figure 1). The index, r', of the node in the linear array A_N which contains the rth key is $s_a + \lceil (r - a^{c-1} + 1)/(b-1) \rceil$ and the offset, k, within that node is $((r - a^{c-1}) \bmod (b-1)) + 1$. Once the index of the node is found, a processor can compute the values i and j in the triple $(\langle i, j\rangle, k)$ [7].

After processor P_r finds the triple $(\langle i, j\rangle, k)$ associated with the key it has to fill, it can determine the rank of the key in the sorted input list using the results from Section 3. In order to determine the rank, each processor needs to compute the set of exponents $E_r = \{a^h,\ a^c,\ a^{c-i},\ a^{i-1},\ b^{h-c},\ b^{i-c},\ b^{h-i}\}$. The set E_r can be computed in $O(\log\log N)$ time. Processor P_r also fills the kth child pointer, $\mathrm{CHILD}_N(\langle i, j\rangle, k)$, using Lemma 3.1. The rightmost child pointer in a node is filled by the processor handling the last key in the node. As it stands, the array A_N represents the desired $(a,\ b)$-tree. However, operations on the $(a,\ b)$-tree may lead to restructuring and the tree may no longer be in a contiguous linear array.

There are no read or write conflicts in the steps outlined above. Hence, the $(a,\ b)$-tree can be constructed with N keys in $O(\log\log N)$ time, using N processors on the EREW PRAM. (If we use the CREW PRAM, the required exponents can be computed in $O(1)$ steps as shown in [15] leading to an $O(1)$-time algorithm.)

Now, Brent's scheduling [4] can be employed to obtain an optimal algorithm on the EREW PRAM. Each processor can handle $O(\log\log N)$ keys in $O(\log\log N)$ time, since the set E_{r+1} can be computed from the set E_r, for any r, in $O(1)$ time. So we have the following.

Theorem 4.1 *The $(a,\ b)$-tree with N keys can be constructed in $O(N/p + \log\log N)$ time, using p processors on an EREW PRAM. This algorithm is optimal for $p \leq N/\log\log N$.*

5 Conclusions

We have presented an optimal parallel algorithm for constructing a minimum-height $(a,\ b)$-tree, with time-complexity independent of a and b. Hence, our general algorithm is superior to the earlier algorithm of Wang, Chen and Yu [16] for constructing B-trees, a special case of $(a,\ b)$-trees. Further, with our algorithm other height-balanced trees, like B*-trees, leaf-oriented $(a,\ b)$-trees, and finger trees can be constructed. See [7] for details.

One measure of the efficiency of any search structure is its memory utilization. We have shown that the average memory utilization of the canonical $(a,\ b)$-tree presented is much better than the worst-case utilization. Moreover, the 2-3 trees and B-trees constructed by our general algorithm have better memory utilization than those constructed by the algorithms given by Wang and Chen [15] and Wang, Chen and Yu [16].

Rosenberg and Snyder [14] have characterized B-trees with the maximum memory utilization. Miller, Pippenger, Rosenberg and Snyder [10] have presented B-trees that have minimal expected number of nodes visited per access. Rosenberg and Snyder [13] have also characterized 2-3 trees that are minimal in expected number of comparisons per access for a given number of keys. Investigation of canonical forms for $(a,\ b)$-trees which are optimal with respect to any of these measures but which can also be constructed fast in parallel is an important topic for further exploration. More work also needs to be done on in-place algorithms for parallel re-initialization of height-balanced trees. Another problem worth investigating is the parallel construction of multidimensional trees, where partial or even global re-initialization is the only means to restore balance [12].

References

[1] R. Bayer and E. McCreight. Organization and maintenance of large ordered indices. *Acta Inf.*, 1:173–189, 1972.

[2] O. Berkman, D. Breslauer, Z. Galil, B. Scheiber, and U. Vishkin. Highly parallelizable problems. In *Proc. 21st Symp. on Theory of Computing*, pages 309–319, 1989.

[3] J. R. Bitner and S.-H. Huang. Key comparison optimal 2-3 trees with maximum utilization. *SIAM J. Comput.*, 10(3):558–570, Aug. 1981.

[4] R. P. Brent. The parallel evaluation of general arithmetic expressions. *J. ACM*, 21(2):201–206, 1974.

[5] H. Chang and S. S. Iyengar. Efficient algorithms to globally balance a binary search tree. *Commun. ACM*, 27(7):695–702, July 1984.

[6] E. Dekel, S. Peng, and S. S. Iyengar. Optimal parallel algorithms for constructing and maintaining a balanced m-way search tree. *Intl. J. Parallel Programming*, 15(6):503–528, 1986.

[7] N. Deo, A. Jain, and M. Medidi. Parallel construction of (a,b)-trees. Technical Report CS-TR-91-21, University of Central Florida, Dec. 1991. Revised Aug. 1992.

[8] P. L. Lehman and S. B. Yao. Efficient locking for concurrent operations on B-Trees. *ACM Trans. Database Syst.*, 6(4):650–670, Dec. 1981.

[9] K. Mehlhorn. *Data Stuctures and Algorithms 1: Sorting and Searching*. Springer-Verlag, Berlin, 1984.

[10] R. E. Miller, N. Pippenger, A. L. Rosenberg, and L. Snyder. Optimal 2,3-trees. *SIAM J. Comput.*, 8(1):42–59, Feb. 1979.

[11] A. Moitra and S. S. Iyengar. A maximally parallel balancing algorithm for obtaining complete balanced binary trees. *IEEE Trans. Comput.*, C-34(6):563–565, June 1985.

[12] M. H. Overmars. *The Design of Dynamic Data Structures, LNCS 156*. Springer-Verlag, Berlin, 1983.

[13] A. L. Rosenberg and L. Snyder. Minimal-comparison 2,3-trees. *SIAM J. Comput.*, 7(4):465–480, Nov. 1978.

[14] A. L. Rosenberg and L. Snyder. Time- and space-optimality in B-trees. *ACM Trans. Database Syst.*, 6(1):174–193, Mar. 1981.

[15] B.-F. Wang and G.-H. Chen. Cost–optimal parallel algorithms for constructing 2–3 trees. *J. Parallel and Distributed Comput.*, 11:257–261, 1991.

[16] B.-F. Wang, G.-H. Chen, and M. Yu. Cost–optimal parallel algorithms for constructing B-trees. In *Proc. 1991 Intl. Conf. Parallel Processing*, pages 294–295, 1991.

[17] B. Zhang and M. Hsu. Unsafe operations in B-Trees. *Acta Inf.*, 26:421–438, 1989.